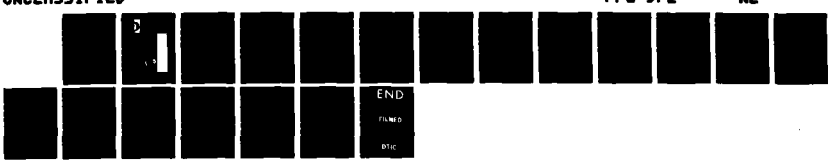
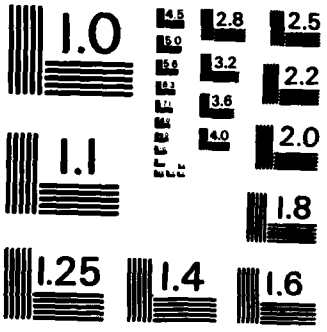
AD-A160 645 A GARBAGE COLLECTOR FOR A LARGE DISTRIBUTED ADDRESS SPACE(U) ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (ENGLAND) S R WISEMAN JUN 85 RSRE-85009 DRIC-BR-97054 1/1

UNCLASSIFIED F/G 9/2 NL





UNLIMITED

DR97054

Report No. 85009

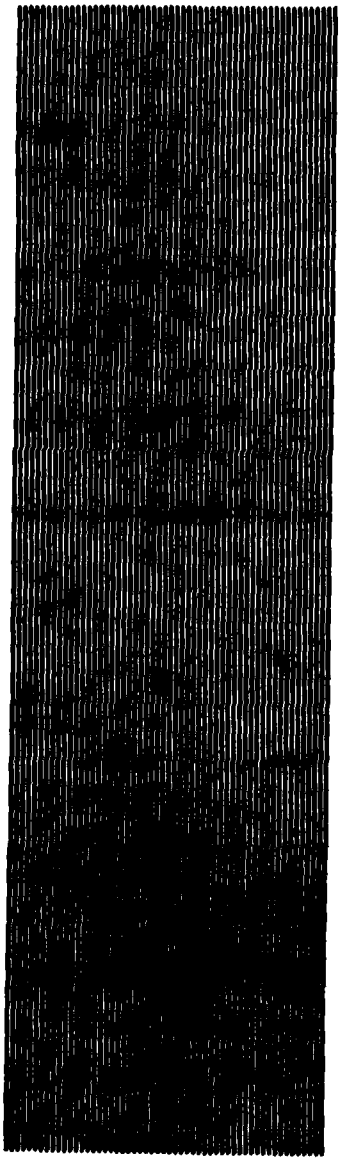
AD-A160 645

Report No. 85009

ROYAL SIGNALS AND RADAR ESTABLISHMENT,
MALVERN

A GARBAGE COLLECTOR FOR A LARGE DISTRIBUTED ADDRESS SPACE

Author: S R Wiseman



PROCUREMENT EXECUTIVE, MINISTRY OF DEFENCE
RSRE
Malvern, Worcestershire.

June 1985

UNLIMITED

UNLIMITED

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Report No 85009

Title: A GARBAGE COLLECTOR FOR A LARGE DISTRIBUTION ADDRESS SPACE

Author: Simon R Wiseman

Date: June 1985

SUMMARY

A new on the fly garbage collection algorithm is presented. It is suitable for use in distributed systems, where the address space is very large. The time taken to recover garbage is reduced by recursively structuring the address space. Consideration is given to the practical implementation of the algorithm.

RSRE REPORT NO 85009

A GARBAGE COLLECTOR FOR A LARGE DISTRIBUTED ADDRESS SPACE

Simon R Wiseman

LIST OF CONTENTS

1 Introduction

2 Block Structured Memories

3 General Garbage Collection Techniques

4 Recursively Organised Block Structured Memory

5 The Garbage Collector

6 Practical Implementation

7 Conclusions

8 Acknowledgements

9 Reference

Appendix A. Algorithm in Pidgin Algol

1. Introduction.

As address spaces get larger, existing garbage collection techniques take longer to recover garbage. Therefore to ensure the free space is never exhausted, greater amounts of spare memory are required. However, a single address space that spans a distributed system will be very large, requiring a very large amount of spare memory.

An important aspect of distributed systems is that they do not offer a perfect environment. Transient faults, such as message loss, and more permanent faults, such as network partitioning, must be tolerated by the garbage collector. Also, the additional costs of communication may cause a severe degradation in performance.

In this paper, a new garbage collection algorithm is proposed. It is intended for use in distributed systems where very large address spaces are encountered. The time taken to recover most garbage is small, allowing a better utilization of memory resources.

2. Block Structured Memories.

A Block Structured Memory is one where the memory is viewed as a set of memory blocks, rather than as one large linear array. Each block is an array of words, where the words are either integers or pointers to blocks. A pointer points to a block as a whole; to address an individual word a pointer and an offset must be supplied. This is illustrated by figure 1.

Block structured memories are found in computers which use capability based addressing [Fabry74], where the pointers are primitive capabilities, in LISP systems [McCarthy60] and Smalltalk systems [Ungar84]. They differ from the more usual segmented memories in the way addressing is performed. A segmented memory places an order on the segments and uses an index, rather than a pointer, to address a segment.

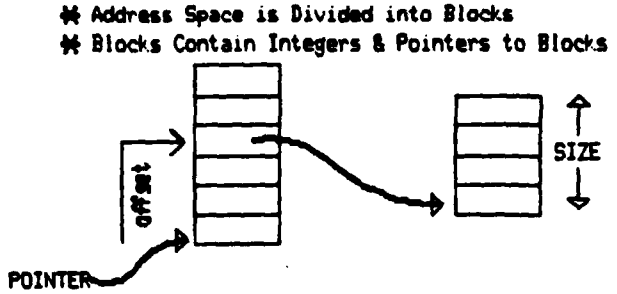


Fig. 1. Blocks and Pointers.

A pointer can be distinguished from a non-pointer. This is done either by tagging, where a bit is used to distinguish between the two possibilities, or by having two kinds of block, one which can contain only pointers and one which can contain only non-pointers. Tagging is used by LISP systems [White80] and the Flex computer [Foster79], while partitioning is used by the Cambridge CAP computer [Needham&Walker77], the Intel iAPX-432 [Tyner81] and the Plessey PP250 [England75].

A pointer contains either the physical address of the block it points to, or an index into a table of physical addresses. The indirection technique allows a block to be moved about in physical memory, since its physical address appears in only one place which allows it to be readily updated, though the indirection imposes some extra run-time overhead..

If blocks can be of various sizes, as in the case of the capability computers and Smalltalk systems, then it must be possible to determine a block's size. This is often done by storing the size in the first word of the block, in which case it is usual for the computer to hide this word from the programmers. An alternative is to store the size with the address in the indirection table.

The registers of the computer can contain pointers or non-pointers. The pointers that are stored in the registers form the root of the accessible structure that is in memory. A block that is pointed to by a pointer in a register is said to be directly accessible. Either tagging or partitioning of the registers can be used to distinguish between pointers and non-pointers stored in them.

Any block that cannot be reached by following a chain of pointers starting at the root can never be accessed. These inaccessible blocks can therefore be recovered and the storage they occupy can be used to make new blocks. It is the responsibility of the garbage collector to identify and recover these inaccessible blocks.

For the purposes of discussing the garbage collection of a block structured memory, the instructions obeyed by the processor will be considered to be made up of five primitive operations. These are:

1. read a word from a directly accessible block and store it into a register;
2. write a word from a register into a directly accessible block;
3. create a new block and store a pointer to it into a register;
4. copy a word from one register into another;
5. load a numeric value into a register.

All these operations may change the structure that is accessible, because they may create new pointers or destroy existing ones. More powerful operations, such as appending an item to a list, are readily constructed from this set of primitives. The primitives also cater for the more mundane arithmetic operations.

3. General Garbage Collection Techniques.

There are many techniques which can be used to recover garbage in a block structured memory. [Cohen81] and [Wiseman85] give comprehensive surveys. There are two ways in which the garbage collector can operate. It can either interrupt the computer's normal processing, and garbage collect the entire memory in one operation, or proceed in pseudo parallel with normal processing, with the garbage collection proceeding gradually.

The former approach can be more efficient in terms of memory cycles, but gives rise to pauses in program execution which could be embarrassing in many applications. The overheads incurred by the latter, "on the fly" approach may be an acceptable price to pay for steady program execution.

There are two main types of on the fly garbage collector, the mark-scan type and the copying type. The mark scan type was first proposed by [McCarthy60] and is formally described by [Dijkstra.et.al.78]. It gradually scans the accessible structure and marks all the blocks that are accessible. The storage occupied by the inaccessible blocks, which are those that are not marked, can then be reclaimed.

The copying type of garbage collector, first proposed by [Hansen69] and refined by [Cheney70], divides the memory into two spaces. Only one space is used at a time. It is garbage collected by copying all the accessible blocks from it into the other space. When all of the accessible blocks have been copied from it the roles of the spaces are reversed. A similar algorithm is used by [Ungar84] in a Smalltalk system.

The advantage of the copying type of garbage collector is that the accessible blocks are inherently compacted to one end of the store. This makes the allocation of new blocks very simple. In contrast, mark scan garbage collectors leave the accessible blocks scattered about in memory, interspersed with the free space. This makes allocation of new blocks more difficult. If the free space becomes severely fragmented, then compaction may be necessary to allow new blocks to be created. This would have to be done as a separate action.

Another technique often proposed for garbage collection is Reference Counting. Here a count is kept with each block of the number of pointers that refer to it. When a pointer is copied the appropriate count is incremented and when a pointer is overwritten the count is decremented. When a block's reference count drops to zero, it is scanned for pointers and the counts of blocks that these point to are decremented. The block is then deallocated.

The main drawback of reference counting is that it cannot recover all garbage. Blocks in cyclic structures which become inaccessible still have positive reference counts and so are not deallocated. For example a block which contains a pointer to itself will never be deallocated because, even if no other pointers to it exist, its reference count will always be at least one.

Maintaining the reference counts can also be very time consuming. Whenever a word is written to memory, and there is a possibility of overwriting a pointer, the memory must first be read. This is so that the the reference count of the block that a doomed pointer refers to can be decremented. This is less of a problem if partitioning is used to identify pointers, though it is a serious performance handicap in tagged systems.

4. Recursively Organised Block Structured Memories.

Existing techniques for garbage collection are inappropriate for a block structured memory that spans a distributed system. One reason is that the memory is likely to be very large. For example it may encompass the main memories of many computers which are networked together. If a garbage collector which suspends normal processing is used, then the pauses will be very long, making the system impractical for most applications. If a mark scan garbage collector is used then the scan will take a long time to complete. During this time it is likely that the available free storage will become exhausted so that program execution will either fail or have to be suspended. Similar problems occur with copying garbage collectors, though there is the additional problem of finding sufficient spare memory.

This paper proposes a way of reducing the time taken to recover garbage using a mark scan garbage collector. This is done by partitioning the memory into disjoint areas. Each area is garbage collected independently, though blocks referenced by other areas require special treatment. If the areas are chosen to exploit the locality of reference that is usually found in distributed systems, then the independent garbage collections will recover significant amounts of inaccessible blocks. Since the areas are much smaller than the whole distributed memory, garbage should be recovered more quickly.

Blocks which are referenced by pointers in other areas must be kept when their area is garbage collected. This could be easily achieved by using one bit to mark the blocks as externally referenced or not. However, when a block ceases to be externally referenced, this bit must be cleared so that the block can be recovered if it becomes garbage. Unfortunately the independent garbage collections of the areas cannot detect when a block ceases to be externally referenced.

The simplest way to maintain the externally referenced marks of the blocks, is to use a mark scan garbage collector that considers all the areas as a whole. This would detect when all external pointers to a block have become inaccessible and would then mark the block as not externally referenced. The garbage collector will cope with inaccessible cyclic structures that span the area boundaries.

The algorithm proposed by this paper goes further than this simple two-level garbage collection. It structures the areas recursively, while still retaining the single address space of the block structured memory. Therefore, an area at one level will consist of several areas at a lower level, and will be part of a higher level area. A block will belong to some area at each level in the hierarchy. Each area will have a garbage collector which is responsible for recovering garbage that spans the boundaries of its component areas.

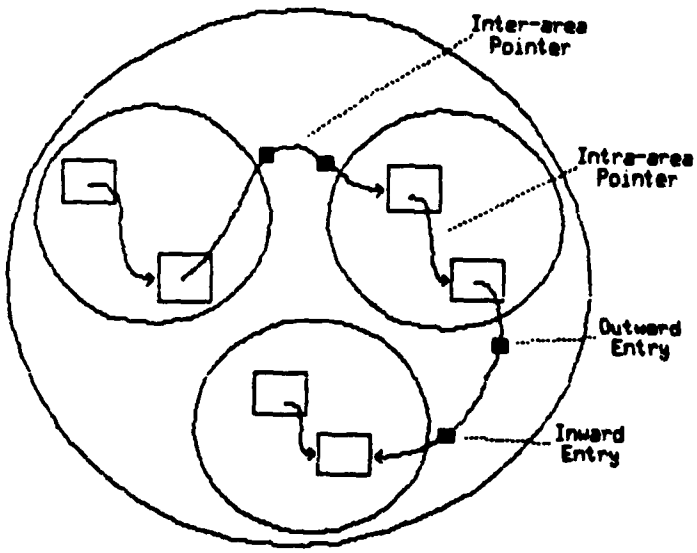


Fig. 2. Address Space is Divided into Areas.

Figure 2 shows a block structured memory divided into three areas. Intra-area pointers can be implemented as direct physical addresses or by indirection. Inter-area pointers are implemented through two indirection entries. The pointer refers to an outward entry. This describes the location of an inward entry which in turn gives the location of the block. The inward entry is essential, as it allows a block to be moved even if it is referred to by another area. This inward entry in fact also takes on the role of the externally referenced bit. The outward entry is however optional, but in a distributed system it is likely that a pointer would not be large enough to hold all the information required to locate the inward entry. Also it allows a useful optimization to be applied, which is described later.

The recursively structured areas are illustrated by figure 3. The inter-area pointers are shown with their outward and inward indirection entries. An inter-area pointer is said to be at level 'n', if the lowest level area that contains both the source and destination of the pointer is level 'n'. The indirection entries for a level 'n' inter-area pointer are shown placed at level 'n'.

The hierarchical mark scan garbage collections are not implemented as separate processes. Instead there is a separate garbage collector process for only the lowest level areas. These are responsible for the garbage collection of their areas, but also contribute to the garbage collection of the higher level areas above them. This means that the amount of scanning and marking is no greater than if one system wide garbage collector was used. Garbage collection of the lowest level areas can proceed as often as required as the processes do not need to be synchronized.

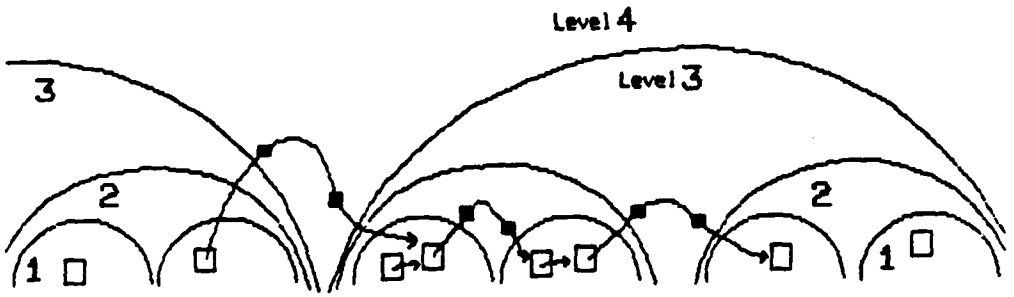


Fig. 3. Recursively Structured Address Space.

5. The Garbage Collector.

1. The Simple Garbage Collector.

The proposed garbage collector is first described in its basic form, ignoring any inter-area pointers. It is similar to that described by [Dijkstra.et.al.78]. All blocks have a field which indicates their state in the scan. The colours white, grey and black are used to represent the states. The garbage collector cycles through three phases; initialization, scanning and recovery. It proceeds in pseudo parallel with the normal operations of the processor, which are represented by the five primitive operations described earlier.

Before initialization all blocks are either white or grey. The initialization phase finds all blocks that are directly accessible, that is those which are pointed to from the registers, and colours them grey. The scanning phase then searches through all the blocks in the area looking for ones that are coloured grey. When a grey block is found it is scanned for pointers. All pointers that are found in it are followed and if the block that is referred to is white then it is made grey, an operation called shading. When all the pointers have been found, the block is coloured black, to indicate that it has been scanned. The scanning phase continues until no more grey blocks remain. The recovery phase then examines all the blocks in the area. White blocks are inaccessible and so are recovered, while black blocks may be accessible and are kept, though their colour is changed to white ready for the next garbage collection cycle.

Since the garbage collector is operating in pseudo parallel with the computer's normal operation, the five primitive actions must ensure that they correctly maintain the colours of the blocks. In the simple case described so far this means that when a pointer is read from store into a register, the block it points to must be shaded. Also newly created blocks, which initially contain no pointers, must be coloured black.

ii. The Final Algorithm.

The complete algorithm for one of the garbage collector processes is now described. This is responsible for the garbage collection of one of the areas at the lowest level, but also contributes to the garbage collection of the higher level areas above it. The algorithm is given in pidgin Algol in Appendix A.

Each BLOCK has a COLOUR field, which indicates its state in the lowest level garbage collection. The colours white, grey and black are used. During the scan white indicates that the block has not been found, grey that the block has been found but still needs scanning for pointers, and black that the block has been found and has been scanned for pointers.

Inter-area POINTERs refer to an OUTWARD entry. This in turn refers to an INWARD entry which describes the location of the BLOCK. The level of the inter-area pointer is given by the level field in both the INWARD and OUTWARD entries. The INWARD entry also has a colour field and a colour_level field. These indicate the colour of the block in the higher level scans.

In the initialization phase, the garbage collector searches the computer's registers, looking for POINTERs. Any pointers in the registers are the roots of the structure in memory. They are therefore considered to be in the highest level area. Intra-area pointers and inter-area pointers that are found in the registers are shaded and externally_shaded respectively. This ensures that the directly accessible blocks are all at least grey.

The garbage collector now works its way down the levels. At each level it looks for blocks that are referenced by other areas and which are grey or black at that level. This is determined by checking all the INWARD entries to find any with colour = grey or black and colour_level = level. These blocks are known to be wanted at this level and so are shaded. The garbage collector now scans all the grey blocks, looking for pointers. Any intra-area pointers are shaded as usual. If an inter-area pointer is found, then it is externally_shaded, but only if the pointer is at a lower or equal level to that being scanned. This is because the pointer may not be accessible at the higher levels.

To ensure that a block is not discarded by the lower levels before it is shaded by a higher level that requires it, it is also necessary to shade blocks referred to by white INWARD entries. However, inter-area pointers accessible from such a block must not be externally_shaded because of this. Therefore scanning from white INWARD entries is delayed by one pass through the levels.

Once the scanning of all the levels has been completed, any BLOCK which has colour = white can be recovered, since it is not accessible from the root.

During the recovery phase of one of the higher levels, any INWARD entry of that level which has colour = white is inaccessible and is discarded. The other INWARD entries of the level have their colour and colour_level reset, ready for the next scanning phase. Since the remaining INWARD entries are now white, the recovery phase must not be entered for a second time, before the next initialization and scanning phases. To ensure this the AREA has a flag associated with it to show whether it has already performed the recovery.

The initialization phase of a higher level garbage collection is needed to ensure that all the lower level areas have completed a scan. This guarantees that all the roots of the area have been externally_shaded. The scanning phase then continues until no inter-area pointers between the lower level areas are coloured grey at, or above, that level. The recovery phase is now entered. This recovers the inaccessible INWARD entries and resets the colour of the remaining INWARD entries. Once all constituent areas of the area have been recovered, the area continues another garbage collection cycle with the initialization phase.

The OUTWARD entries are treated very much like blocks. They have a colour field which shows their state in the lowest level garbage collection. However, since they are not blocks which can contain pointers, they are coloured black immediately. Any OUTWARD entries with colour = white during the recovery phase are discarded, because no accessible pointers point through them.

iii An Example Garbage Collection.

To illustrate the garbage collector in action, consider figure 4. This shows five areas, A..E, grouped together into two level-two areas, R and S. These in turn make up a level-three area, T, which is the entire address space. The blocks that are of concern are numbered 1..6. The root of the accessible structure is the pointers that are contained in the registers of the computer. These are considered to be at level-four. In this example the root is a pointer to block 4. Four pointers, W..Z, at various levels are shown. Initially all the INWARD entries have colour = white. Figure 5 shows how these values change as the garbage collection proceeds.

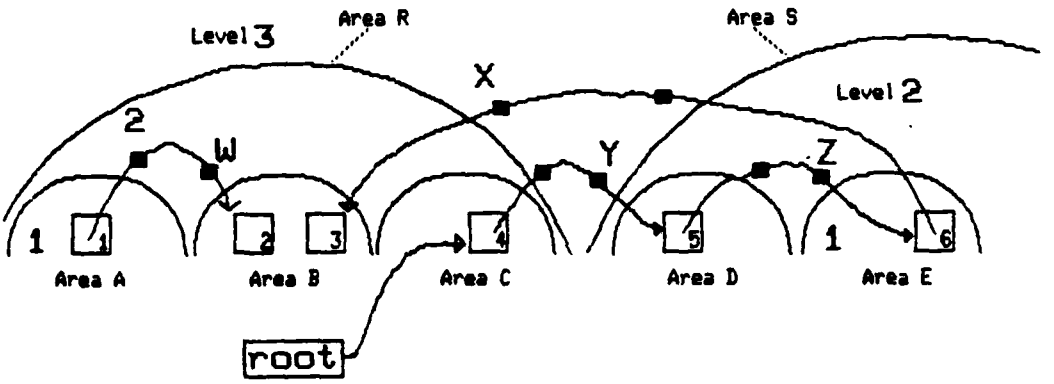


Fig. 4. The Initial State for the Example Garbage Collection.

	W	X	Y	Z
Initially:	W	W	W	W
Garbage Collect C:	--	--	4G	--
Garbage Collect D:	--	--	4B	4G
Garbage Collect B:	--	--	--	--
Garbage Collect A:	--	--	--	--
Recover R:	discard	--	--	--
Garbage Collect E:		4G	--	4B
Recover S:		--	--	W
Garbage Collect B:		4B	--	--
Recover T:		W	W	--

Fig. 5. States of the Inward Entries during the Example Garbage Collection.

First suppose that the garbage collection process for area C performs a cycle. The pointer to block 4 would be found in the root and the block would be shaded. The scan would then find the outgoing pointer Y. This would be externally_shaded at the level of the scan, which is level 4. Therefore the INWARD entry of Y changes from white (W) to level 4 and grey (4G). Note that the OUTWARD entry of Y was coloured black by external_shade. This ensures that it is kept during the recovery phase of area C.

Now suppose that area D performs a garbage collection cycle. The level 4 scan finds the INWARD entry for pointer Y, because it is now 4G. Therefore block 5 is shaded. The INWARD entry is made 4B to indicate that it has been scanned. The level 4 scan proceeds and the outgoing pointer Z is found. The OUTWARD entry for Z is made black and the INWARD entry is made 4G.

If area B now performs a garbage collection, then its level 4 scan will find nothing. This is because no roots point into the area and no INWARD entries are 4G or 4B. Similarly the level 3 scan does nothing, because the INWARD entry for X is white, and so is not considered until the level 2 scan. The level 2 scan does find X, and so block 3 is shaded and is therefore will not be discarded. The level 2 scan does not consider pointer W because it is white. This is found by the level 1 scan, so block 2 is also kept.

Block 1 in area A is not accessible. The garbage collection of area A will therefore not shade block 1 and it will be discarded. Similarly the OUTWARD entry of pointer W will not be found and it will be discarded. The INWARD entry of W will however remain, since it may still be used by other OUTWARD entries.

Now that areas A, B and C have each completed a garbage collection cycle, it is possible for area R to proceed from the initialization phase to the scanning phase. However, since there are no grey INWARD entries coming into the area, it can immediately proceed to the recovery phase. This will consist of the three areas A..C performing another garbage collection. Only that of area B is of interest, so that will be explained in more detail.

The garbage collection of area B, with area R in the recovery phase, starts off with a level 4 scan and a level 3 scan which do nothing. The level 2 scan proceeds differently because the level 2 area is in the recovery phase. The level 2 INWARD entries are examined. White entries are discarded, and black entries are reset to white. Therefore the INWARD entry for pointer W is discarded, which completes the destruction of the inaccessible pointer. Then the INWARD entry for pointer X is found, because it is white and in the next highest level. Block 3 is shaded and will therefore eventually be kept. After the level 1 scan, block 2 remains white and so it is discarded.

Now suppose area E does a garbage collection. The level 4 scan finds the INWARD entry for pointer Z, and so it is changed from 4G to 4B and block 6 is shaded. The level 4 scan then finds the pointer X in block 6, so the INWARD entry of X is made 4G.

Area S has now completed its initialization phase, and also its scanning phase. During the recovery phase Z is returned to white. Now that both areas R and S have completed their scan phases, area T can enter the scanning phase. However, because the pointer X is still grey area T has not completed its scan phase. However, once area B has completed another garbage collection, X will have become 4B, so area T enters the recovery phase.

During the recovery phase of area T, the pointers X and Y are reset to white. This completes a cycle of the garbage collection of the whole system. The initial condition is restored, except that blocks 1 and 2 have been discarded.

6. Practical Implementation.

In a practical implementation of this algorithm, the memories of each computer in the distributed system would probably be a separate area. The next level of area might be a local area network which connects together a group of computers. Higher levels could be an inter-network of the local networks. However, it is also possible to have the area which is one computer's memory composed of smaller areas. This partitioning of the memory may be useful in controlling the allocation of memory and garbage collection time in a time-sharing system.

Pointers which point between computers will have INWARD and OUTWARD indirection entries. The OUTWARD entry would exist in the memory which contains the pointer and the INWARD entry would exist in the memory which contains the block. The operation of external_shading therefore requires a communications protocol between the two computers. Since this is likely to be unreliable, it should be noted that the shading operation can be repeated without adverse affect.

The amount of traffic generated by external_shading can be reduced by using the OUTWARD entry to remember the highest shade sent through it. If a pointer is found more than once during a scan, then the shade need only be sent once. The OUTWARD entry would be reset during the recovery phase of the appropriate level.

Another optimization that can be performed with OUTWARD entries, is reference counting. If the INWARD entries keep a count of the number of OUTWARD entries which refer to them, then if the count drops to zero the INWARD entry can be discarded. Whenever a new OUTWARD entry is created, the count must be increased. This will occur when a pointer is copied from one area to another. The incrementing must be performed by the originator of the pointer, and must complete before the originator is allowed to decrement the count. Whenever an area discards an OUTWARD entry, it can decrement the count in the INWARD entry. This requires a communications protocol. Note that decrementing a count must never be retried, in case it is actually performed more than once. However, incrementing a count can be retried. If the reference count does become too high, because of communications failures, then the INWARD entry will be recovered using the normal scanning action of the garbage collector.

The garbage collectors need to agree when to change the phase of the higher levels of garbage collection. This requires a communications protocol which is tolerant of many types of failure. The simplest form of protocol would be polling the various areas to see if they are ready. When a complete circuit is made without finding any dissenters, then the next phase can begin. This is similar to the algorithm given by [Dijkstra.et.al.78]. However, this simple protocol is prone to node crashes and network partitioning.

The correct operation of the garbage collector relies on the participants correctly obeying the communication's protocols. For example an area could crash the whole system by placing an incorrect INWARD entry identifier in a request. We must therefore be sure that the garbage collector is correct. However, if the network is open to other systems then these too could conceivably crash the distributed system by forging a transaction. This is the general problem of authentication on an open network, which is a research topic beyond the scope of this paper.

Some implementations of block structured memories have a kind of pointer which does not protect the block they refer to from garbage collection. Some LISP systems call them weak, as opposed to strong, pointers, and the Flex computer calls them shaky, as opposed to firm, pointers. If the garbage collector finds a block which is only referred to by weak pointers, then the block is discarded and all the pointers that refer to the block are made into the nil pointer. This can be achieved in a scanning garbage collector by introducing another colour, say silver. If a weak pointer is found during the scan, then the block is weakly shaded. This causes white blocks to become silver. Silver blocks are not scanned. During recovery, silver blocks are discarded, but a tombstone [Lomet75] must be erected in place of the block, so that the weak pointers may be set to nil during the next scan.

The algorithm given in Appendix A, does not show the critical sections that are required to ensure the correct operation of the garbage collector. This is because the algorithm is intended to be used on a very coarse basis. For example the computer may have a garbage collection instruction which would perform some garbage collection for a fixed amount of time. This would guarantee that no list processing operations are being performed while the garbage collector was operating. It is however necessary to specify safe places at which the garbage collector may suspend its operation. The only difficult requirement is that pointers stored in the registers must never point to white blocks. This could occur if the garbage collector was suspended during recovery. Therefore if this happens it is necesary to shade all pointers stored in registers before continuing with list processing.

7. Conclusions.

This paper has presented an algorithm which allows the timely garbage collection of a large block structured memory. This makes it suitable for use in a distributed system which has one very large address space. More research is required to develop suitable communications protocols to allow the algorithm to function correctly in a faulty environment. The problem of operating in an open communications system also needs to be addressed.

8. Acknowlegements.

Thanks go to Prof. Brian Randell of the University of Newcastle upon Tyne and Dr. Derek Barnes of RSRE for their helpful comments on earlier drafts of this paper.

9. References.

C.J.Cheney
A Nonrecursive List Compacting Algorithm.
Comms. of the ACM
Vol 13, Num 11, Nov 1970, pp677..678

J.Cohen
Garbage Collection of Linked Data Structures.
ACM Computing Surveys
Vol 13, Num 3, Sept 1981, pp341..367

E.W.Dijkstra, L.Lamport, A.J.Martin, C.S.Scholten & E.F.M.Steffens
On-the-Fly Garbagae Collection: An Exercise in Cooperation.
Comms. of the ACM
Vol 21, Num 11, Nov 1978, pp966..975

D.M.England
Capability Concept Mechanisms and Structure in System 250.
Rev. Fr. Autom. Inf. Rech. Oper. (France)
Vol 9, Sept 75, pp47..62

R.S.Fabry
Capability Based Addressing.
Comms. of the ACM
Vol 19, July 1974, pp403..412

J.M.Foster, C.I.Moir, I.F.Currie, J.A.McDermid, P.W.Edwards, J.D.Morison & C.H.Pygott
An Introduction to the Flex Computer System.
RSRE Report 79016
Oct 1979

W.J.Hansen
Compact List Representation: Definition, Garbage Collection, and System Implementation.
Comms. of the ACM
Vol 12, Num 9, Sept 1969, pp499..507

D.B.Lomet
Scheme for Invalidating References to Freed Storage.
IBM Journal of Research and Development
Jan 1975

J.McCarthy
Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part 1.
Comms. of the ACM
Vol 3, 1960, pp181..195

R.M.Needham & R.D.H.Walker
The Cambridge CAP Computer and its Protection System.
Operating System Reviews
Vol 11, Num 5, Nov 77, pp1..10

P.Tyner
iAPX-432 General Purpose Data Processor: Architecture Reference Manual.
Intel Corp. Jan 1981

D.Ungar
Generation Scavenging: A Non-disruptive high Performance Storage Reclamation Algorithm.
ACM SigPlan Notices
Vol 19, Num 5, May 1984, pp157..167

J.L.White
Address/Memory Management for a Gigantic LISP Environment, or GC Considered Harmful.
Procs. LISP 80 Conference
July 1980, pp119..127

S.R.Wiseman
On the Garbage Collection of Block Structured Memories.
RSRE Report 85006.
May 1985.

APPENDIX A. Algorithm in Pidgin Algol.

```
TYPE
   COLOUR  = (white,grey,black),
   PHASE   = (init,scan,recover),
   AREA    = STRUCT( PHASE phase, BOOL ready, ..... ),
   BLOCK   = STRUCT( COLOUR colour, ........ ),
   INWARD  = STRUCT( INT level, REF BLOCK block,
                     COLOUR colour, INT colour_level ),
   OUTWARD = STRUCT( COLOUR colour, INT level, REF INWARD inward, ..... ),
   POINTER = UNION( REF BLOCK intra, REF OUTWARD inter );

[2:max] AREA area;

PROC shade( BLOCK block ):
BEGIN
   IF block.colour = white THEN block.colour := grey FI
END;

PROC external_shade( OUTWARD inter, INT level ):
BEGIN
   IF inter.colour = white THEN inter.colour := black FI;
   IF inter.level <= level
   THEN
      IF inter.inward.colour_level < level
      OR ( inter.inward.colour_level = level AND inter.inward.colour = white )
      THEN
         inter.inward.colour := grey; inter.inward.colour_level := level
      FI
   FI
END;

PROC scan( INT level ):
BEGIN
   FOR each grey block
   DO FOR each pointer in the block
      DO IF pointer IS intra
         THEN shade( pointer.intra )
         ELIF level >= pointer.inter.level
         THEN external_shade( pointer.inter, level )
         FI
      OD;
      block.colour := black
   OD
END;

PROC scan_done( INT lower_level ):
BEGIN
   INT level = lower_level + 1;
   IF area[ level ].phase = init
   THEN
      IF all others are ready
      THEN area[ level ].ready := false;
           area[ level ].phase := scan
      ELSE area[ level ].ready := true
      FI
   FI;
```

```
   IF area[ level ].phase = scan
   THEN
      IF no inward entry has colour = grey and colour_level >= level
      THEN
         IF all others are ready
         THEN area[ level ].phase := recover;
              area[ level ].ready := false;
              scan_done( level )
         ELSE area[ level ].ready := true
         FI
      FI
   ELIF area[ level ].phase = recover
   THEN
      IF all others are ready
      THEN area[ level ].phase := init;
           area[ level ].ready := false
      ELSE area[ level ].ready := true
      FI
   FI
END;

{ initialize }
FOR each register containing a pointer
DO IF pointer IS intra THEN shade( pointer.intra )
                       ELSE external_shade( pointer.inter, max_int ) FI
OD;
{ scan }
scan( max_int );
FOR level := UPB area STEP -1 UNTIL 1
DO IF area[ level ].phase = recover
   AND NOT area[ level ].ready
   THEN
      FOR each inward entry with inward.level = level
      DO IF inward.colour = white THEN discard( inward )
                                  ELSE inward.colour := white FI
      OD
   FI;
   FOR each inward entry
   DO IF inward.colour = black AND inward.colour_level = level
      THEN shade( inward.block )
      ELIF inward.colour = grey AND inward.colour_level = level
      THEN shade( inward.block );
           inward.colour := black;
           area[ level ].ready := false
      ELIF inward.colour = white AND inward.level = level + 1
      THEN shade( inward.block )
      FI
   OD;
   scan( level )
OD;
scan_done( 1 );
{ recover }
FOR each block and outward entry
DO IF block.colour = white THEN discard( block )
                           ELSE block.colour := white FI
OD.
```

DOCUMENT CONTROL SHEET

Overall security classification of sheet ..... UNCLASSIFIED .....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference REPORT 85009	3. Agency Reference	4. Report Security Classification UNLIMITED
5. Originator's Code (if known)	6. Originator (Corporate Author) Name and Location ROYAL SIGNALS AND RADAR ESTABLISHMENT		
5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location		

7. Title
A GARBAGE COLLECTOR FOR A LARGE DISTRIBUTED ADDRESS SPACE

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials	9(a) Author 2	9(b) Authors 3,4...	10. Date	pp. ref.
WISEMAN S R				

11. Contract Number	12. Period	13. Project	14. Other Reference

15. Distribution statement
UNLIMITED

Descriptors (or keywords)

continue on separate piece of paper

Abstract

An on the fly garbage collector for a distributed block structured address space is described. The time taken to recover reasonable amounts of garbage is reduced by recursively structuring the distributed system. This makes it practical to consider a distributed computer system with a single address space.

S80/48

END

FILMED

12-85

DTIC